

**HIERARCHICAL CONNECTED GRAPH MODEL FOR
IMPLEMENTATION OF EVENT MANAGEMENT DESIGN**

A portion of the disclosure of this patent document contains material which is subject to
5 copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone
of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office
patent files or records, but otherwise reserves all copyright rights whatsoever.

CROSS-REFERENCE TO RELATED APPLICATION

This patent application is related to and contains common disclosure with co-pending and
10 commonly assigned patent application "System, Method, and Protocol for Automated
Implementation of Event Management and Monitoring System", serial number 09/488,689, filed
January 20, 2000. The copending patent application is hereby incorporated by reference into this
description as fully as if here represented in full.

TECHNICAL FIELD

15 The present invention is generally related to computer systems management. More
particularly, the present invention relates to a method and system for the automated implementation
of event management design and monitoring for a distributed computing environment.

BACKGROUND

System and network problems are complex and can surface in many different forms. Even a simple problem can leave behind a complex event trail. This can result in excessive confusion in problem determination efforts and unnecessary down time. However, in a well-tuned event management environment, event messages from sources such as application software, database monitors, system hardware and network management systems can often be linked together, or "correlated," to identify the root cause of a problem. Correlation minimizes resources required to resolve the problem and reduce the resulting cost incurred.

The system and network management software industry is focusing increased attention on products that offer support for the disciplines of distributed monitoring and event management. All major systems management vendors are aggressively marketing products that promise improved capabilities in the area of enterprise systems management. There are many products that are currently available in the market. Products include Tivoli Systems' Tivoli Enterprise Console (TEC) and NetView, Computer Associates' CA UniCenter, Hewlett-Packard's OpenView and Event Correlation Service, BMC's Software Patrol, CommandPost and MAXM products and VERITAS' NerveCenter Pro. The invention is described in the context of the Tivoli TEC as an exemplary platform, but the invention is not limited to the TEC platform. Those skilled in the art will realize that many modifications and adaptations to the present invention are possible, and may even be desirable, in certain circumstances and are considered part of the present invention.

While computer networking customers may choose a broad range of event management product alternatives, there has been virtually no attention given by vendors of event management and monitoring products to develop methodologies and tools that assist the customers in effectively relating these event management and monitoring technologies to the specifics of their systems and distributed computing environments. As the individual customer's systems and distributed computing environment typically generate at least hundreds, if not thousands, of unique event messages, without systematic approaches to the tailored implementation of event management, and monitoring technologies, the resulting value to the user of these products is significantly reduced.

For the foregoing reasons, there is a need for a system and method that bridge the gap between the customer's unique systems and distributed computing event environments and the capabilities of an event management product.

SUMMARY OF THE INVENTION

This invention describes a set of linked activities and supporting automation tools that provide for the analysis and documentation of a customer's systems and distributed computing network monitoring and event processing requirements. This methodology is designed as a front-end to effective implementation of event management products such as Tivoli Enterprise Console (TEC), which is a platform described herein for exemplary purposes only. This methodology is open, i.e., capable of front-ending the implementation of any distributed monitoring or event management product, and the invention is not limited to any specific products. The output of the methodology includes a set of design documents that serve as input to effective customization of monitoring and event processing facilities and the development of complex event correlation rules. The methodology is supported by a set of personal computer-based analysis and documentation tools. This specification also describes a software implementation of the result of such methodology, thus preventing an ad hoc approach by individual implementers.

Event Management Design (EMD) is a process developed by IBM Corporation to define the policies and procedures for managing the flow of events in a distributed computing environment. This encompasses the identification of events, filtering practices, and correlation analysis. The process is product independent. At the end of the design, the client is presented with a set of spreadsheets and graphical representations for every source of events in their particular environment. The spreadsheets contain the listings of all events, their enterprise significance, the event significance, and their necessity for correlational analysis. The resulting drawings, using a product such as Visio, are graphic representations of the correlations. These two documents provide all the necessary information for implementing the design as a practical solution.

IBM has further extended EMD to include a design implementation as described in co-pending patent application 09/488,689 which is incorporated by reference herein. This invention provides extensions to the functionality of the tools used in EMD. It includes a spreadsheet that is used to aid in the development of Basic Recording of Objects in C (BAROC) files. Once certain detailed information is added to this sheet, it automatically builds the BAROC files, using whatever class names are provided. The drawing diagrams include code that allows for the generation of Tivoli Enterprise Console (TEC) rules, using specified basic rule templates developed by the EMD implementor. The new drawing code propagates the templates with appropriate class names as determined from the drawings, as well as adding in status and severity changes, timing delays, and locations of scripts to be executed.

There is a single drawing file for each type of unique event source (e.g., hardware device, operating system platform, application). The entire suite of correlation drawings define an Event Relationship Network (ERN). The ERN includes a series of pages which show the subset of correlation relationships. Each of these pages is called a subnet. Subnets may link to other subnets, spanning as many pages as required to fully represent the set of correlational relationships. To a certain extent, the set of events on a given subnet is somewhat arbitrary. There is a physical limitation to the number of events that can be placed on any given page, and the break from page to page is the decision of the implementor. On the other hand, a subnet may contain a complete logical set of relationships, especially when it does not span to any other pages.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates an example of a correlation diagram having a single event with a clearing event.

Fig. 2 illustrates an example of a correlation diagram having a single chain of events with clearing events.

Fig. 3 illustrates an example of a complex correlation diagram.

Fig. 4 illustrates a subnet descendant class structure of the present invention.

Fig. 5 illustrates a subnet ancestor class structure of the present invention.

Fig. 6 illustrates the complex correlation diagram of Fig. 3 with path identification numbers added.

Fig. 7 illustrates the complex correlation diagram of Fig. 3 with path identification and sequence identification numbers added.

Fig. 8 illustrates the processing logic to define an Event Relationship Network (ERN) in accordance with an exemplary embodiment of the present invention.

Fig. 9 illustrates the processing logic for generation of BAROC classes in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

The Tivoli Enterprise Console is used in the following description as the operating platform to facilitate understanding of the present invention. Tivoli's Enterprise Console is a powerful event management and automation application that provides rules-based event correlation for various system components including network, hardware, database, and applications. It offers a centralized, global view of a distributed computing enterprise, while ensuring high availability of applications and computing resources. The Enterprise Console has three primary features that distinguish it from other products: event integration, event processing, and event response and notification. The Enterprise Console receives, processes, and automatically responds to system events, such as a database server being down, a network connection being lost, or a batch processing job completing successfully. It acts as a central collection point for alarms and events from a variety of sources.

In a classic Tivoli implementation, the relationship of each event in the correlational chain with each and every other event in the chain has to be considered. As such, a simple information function can be defined that determines the number of relationships that must be known and implemented for any given chain of events. If $I(fn)n$ represents the information function for a set of n related events, then a classic implementation requires that the total number of states to be known is:

$$I(fn)n = n(n-1)$$

However, the event relationship can be treated not as a linear flow of events, but rather as a hierarchical connected graph. These graphs are defined as hierarchical since any event potentially contains (implicitly) the information of all the preceding events. If this were not the case, then correlational analysis could not be performed since, by definition, the events would share no common information, and would then be random, unrelated occurrences. By examining the set of events as a connected graph, the amount of information required to understand an individual event's relationship to any other event in its connected network can be reduced. If n_i represents an element in the chain, and p_i represents the number of paths the element is connected to within the graph, and if $s = 1$ is defined to be the sequence position along the graph, and by default a single condition of information, then the total number of conditions that need to be known for any element is:

$$I(fn)n_i = p_i + s$$

$$= p_i + 1$$

For a chain of x events, the total number of states can be defined to be:

$$\sum I(fn)n_i \text{ for } i=1 \text{ to } x$$

If a chain of events is considered, starting with a single root cause event, which then has two branches with each branch containing four resultant events, then the following results illustrate the advantage of the present invention:

1. For a traditional Tivoli implementation, the total number of states that need to be known is:

$$I(fn)n = n(n-1).$$

$$= 9(8)$$

$$= 72.$$

In the case of a hierarchical connected graph, the number of states that need to be known is:

$$\sum I(fn)n \text{ for } i=1 \text{ to } x$$
$$= 3 + 2 + 2 + 2 + 2 + 2 + 2 + 2 + 2$$
$$= 21.$$

5 The first element represents the root cause (two branches plus one positional information equals three states). Since all other events only exist on a single path, they each have one branch plus one position equals two states. The propagation of this path and sequence information is done automatically by an EMD/IT tool, based on the constructs described herein.

10 Of the two driving purposes of TEC monitoring, i.e., 1) correlation to determine root cause, and 2) automated response to events, only the correlation problem is considered in this invention. In the present context, correlation analysis is defined as establishing a set of relationships where the ultimate causal event can be determined. This is the primary purpose of the TEC logic engine. Automated response to a causal event is not inherently built into the TEC code, but rather is the use of appropriate scripts executed in response to the determination of the causal event. As such, the scripts are external to the TEC code, and may vary among different implementations, depending on the system management process. The development of automated scripts is a separate process-driven function from correlation analysis.

The key elements of the present invention are as follows:

1. definition of events based on a connected graph model;
2. hierarchical naming convention to be used in the BAROC (Class Definition) files;
3. informational slot definitions required to effectively perform a correlation (root
5 cause) analysis;
4. a basic ruleset template that handles all correlations; and
5. modifications to the software toolset described in co-pending patent application,
serial number 09/488,689 to automate the correlation process.

10 The present invention has a number of advantages as described in this paragraph. BAROC
file class definition structure and naming structure are standardized. The naming conventions and
structure are integrated into the logical design resulting from EMD. The logical flow of event
correlation is totally reflected in the BAROC files. Any support person can now work through the
logical structure without having access to the original EMD material. Path and sequence
15 identification (ID) searches to determine event status is more efficient than current methods permit.
It reduces the requirement to search the entire event cache for multiple events. By integrating the
EMD results, BAROC files and rule sets, a system is created that lends itself to well-governed
change management procedures. No changes can be made on any single component without
affecting the others. As such, any change will require a comprehensive review of the logical
20 structure, and the implications for the implementation. Documentation of changes is a requisite to
maintain the integrity of the integration. By reducing to six the number of essential templates,
implementation can be faster, and the skill level required to implement is reduced accordingly. A
rulebase and BAROC structure are created that are consistent across the enterprise.

In order to make the most efficient use of the logic engine, only one event within a given set of causal relationships can be available for correlation at any given time. All other events will have some aspect of their status changed to remove them from future correlation analysis. There are many ways in which this can be achieved. The event can be dropped, closed, its severity changed, or a new slot can be created for all events which can be flagged for its status as a correlation candidate. The ultimate goal is that at any given time, a console can be viewed and the only events appearing would be the best estimate of a causal event at that given time.

The description of the logical flow of events in a given system makes use of the following definitions:

Autonomous Events: These are isolated events. In other words, they can have no causal events, nor can they be a proximate cause for any other event. There can also be no clearing events for autonomous events. As such, these events will never appear in a Visio diagram, as there is no logic flow associated with their existence. All autonomous events are handled by a single rule as defined by policy, i.e., duplicate detection, escalation, trouble ticketing, etc.

Primary Event: This can also be defined as the root cause event. It can have no precedent events, but may have any number of ancillary events as a result of its existence.

Primary/Secondary Event: This event can be either a causal event, or can be the result of some other event distal to the root cause event.

Secondary Event: This event can only result from some other event distal to the root cause event. It can never occur spontaneously, nor can it be the causal event for any other event.

Clearing Event: This event signals a return to some defined steady state or normal status. A single clearing event can clear multiple events. A clearing event can never have a causal precedent.

Subnet: A subnet is the set of events with appropriate logical flow completely diagrammed. It is represented by a single page within an ERN. A subnet can be autonomous, causal, or secondary to other subnets, and by inference, other events.

Connector: The connector identifies the direction of logical flow.

Using these definitions we can now redefine the several types of events as follows:

Autonomous Event: has no connectors associated with it.

Primary Event (P): can have n connectors flowing away from it; must have zero connectors flowing into it, with the exception of a connector from a clearing event.

Primary/Secondary Event (P/S): can have n connectors flowing into it; can have m connectors flowing out of it.

Secondary Event (S): can have n connectors flowing into it; must have zero connectors flowing out from it.

Clearing Event (C): can have n connectors flowing out from it; must have zero connectors flowing into it.

Subnets can be defined under the same requirements. As such, there can be autonomous subnets, primary subnets, primary/secondary subnets, or secondary subnets. There is no such thing as a clearing subnet.

Fig. 1 illustrates an example of a single event with a clearing event. This is probably the simplest type of correlation possible. A single event occurs, and since it is neither caused by any other event, nor does it cause any other event, it is by definition a primary event. The only event associated with it is a clearing event that signals a return to normal events.

Fig. 2 illustrates an example of a single chain of events with clearing events. This represents a more typical system of correlation, where there is a single chain of causal events, and a set of explicitly defined clearing events. It should be noted that the event that clears both the primary/secondary event and the secondary event is normally represented in Visio as only clearing the primary/secondary event, since clearing of the secondary event is implied by the logical event flow. The explicit clearing in this case is a requirement under the automation system currently used by the EMD implementation in Tivoli Enterprise Console (TEC). The above system better expresses the definition of a subnet, where there is a complete set of events and logical flow designed. A point to consider, which is described further below, is that only the causal chain of events are required for correlation analysis, since the clearing events are restricted to the local subnet. As such, the concept of a path or paths of causal events within a subnet is important to bear in mind.

Fig. 3 illustrates an example of a complex correlation diagram. Intrinsic to this is the internal subnet, which consists of a single primary event, multiple primary/secondary and secondary events. Subnet A represents a causal subnet to one chain of logical flow, while Subnet B is a secondary subnet resultant from a single primary/secondary event. The logical flow illustrates the possibility of multiple paths from any particular event, as well as the possibility for multiple flow into any given event. This example provides all the necessary complexity to illustrate the principles for event management design of the present invention.

The necessary and sufficient conditions for correlation analysis are as follows:

1. the position of any event on a logical directed graph can be defined by two terms – path identification and sequence identification;
2. since the events may belong to multiple paths, the path identification must be a list of integers, conceptually, this is the set of all paths that flow through any given event;
3. the sequence identification is a single integer that defines the relative position of the event on the path or paths.

Events can then be redefined in the following expressions of path ID and sequence ID:

Primary Event: Path ID = $\{1,2,3,...n\}$; Sequence ID = 0. Since the event is the root cause, it can be defined as its own class, and all other events flowing from it become a subclass of it. This accelerates the search process.

Primary/Secondary Event: Path ID = $\{1,2,3,...n\}$; Sequence ID $\in \{n, n+1,...m\}$.

Secondary Event: Path ID = $\{1,2,3,...n\}$; Sequence ID = maximum integer value for any of the paths of which it is a member.

Clearing Event: Path ID = $\{-1,-2,-3,...-n\}$; Sequence ID = 0. The negative path ID is required to signal that this is a clearing event, and not a candidate for root cause correlation.

Implicit to this is that any event that is cleared by this event has knowledge of the negative path ID associated with it.

Therefore, the only necessary condition required to determine the relative causal status of an event is as follows:

Step 1: Is the event in the path of a known event?

If yes, go to step 2.

If no, the event is the current primary event.

Step 2: What is the sequence ID of the event?

If the event sequence number is greater than the known event in the path, then the event is a secondary event, and is removed from further correlational analysis. If the event sequence number is less than the known event in the path, then the event is the current primary event, and the previous known event is removed from further correlational analysis. It is necessary at this time to reiterate that only one event within a given path will remain open for correlation at any given time.

The above-described meta-algorithm is sufficient to determine the status (i.e., primary or secondary) of any event within a given subnet. A problem occurs when multiple subnets are introduced. Logically, the above algorithm can be extended through all connected subnets, but the problem becomes one of propagating all the path IDs in a consistent manner. This is not a true NP-hard problem, but is bounded by a very large polynomial time signature. However, by synthesizing the subnet names with the BAROC class definitions, the search process can be minimized to class structure only.

Unfortunately, simply identifying a subnet as a BAROC class is not sufficient. It is not inconceivable, and is actually probable, that only one path within a subnet leads to another subnet, and there may be other events within the primary subnet that should not be correlated with the secondary subnet. If all events in that subnet were events defined as descendant to the metaclass named for the subnet, then the process would result in spurious correlations. As such, a structure for BAROC class names and hierarchy needs to be defined that reflects the logical flow of events. The following is an exemplary structure for BAROC class naming.

ERN_Class: All events within an event source (Lotus Notes, AIX, Cisco) become members of this class.

ERN_Autonomous: All autonomous events go in this class, since they are handled by a single rule that handles issues like duplicate detection, trouble ticketing, etc. This is a descendant class of Event_Class.

Subnet Class: All events in the subnet become members of this class. It is also a descendant class of ERN_Class.

Subnet Clearing: All clearing events are in this class. This is a descendant class of ERN_Class.

If the subnet has no connections to other subnets, then this is the finest resolution necessary.

All internal correlations are dependent on path ID and sequence ID, so there will be no searching of classes beyond the Subnet_Class level. If the subnet does connect to other subnets, then the following class structures must be incorporated:

Subnet_Descendant#: If elements of the subnet are secondary to events on another subnet, then all events in the direct flow from the primary subnet entry point are members of this class. It is a descendant class of ERN_Class. The # value is merely indicative that there may be multiple flows within the subnet that need to be considered as separate classes. The subnet descendant class structure is illustrated in Fig. 4.

Subnet_Ancessor#: This is the logical inverse of Subnet_Descendant. However, it poses some unique problems. The general definition is that if elements of a subnet are primary to another subnet, then those elements are a member of the class Subnet_Ancessor. These may

be either descendant to ERN_Class or Subnet_Descendant, depending on the logical structure. Unlike Subnet_Descendant, not everything in the flow can be placed into one class. The problem comes when there are branches in the logical flow that lead to multiple subnets. In order to make class naming discrete, the following structure must be used. A nodal event is defined as one that has n paths flowing out of it. A proximal nodal event is one that is closer to the root cause event of the entire path, while a distal nodal event is one that is farther away. The terminal event is the event where the path flows to the next subnet, and is essentially a specialized distal nodal event. Events are then clustered into Subnet_Ancessor classes by following a simple rule. All events assigned to a specific Subnet_Ancessor are from a distal nodal event to the next most proximal nodal event. However, the cluster does not include that proximal event. The subnet ancestor class structure is illustrated in Fig. 5.

The correlation template described in co-pending patent application, serial no. 09/488,689 is modified as described below:

1. The following slots need to be defined in a BAROC file immediately descendant from root.baroc (the ancestral class definition for all events): path_id, sequence_id, descendant_class, and ancestral_class. The path_id will be a list of integers, the sequence_id is a single integer, and the descendant_class and ancestral_class are both a list of string names as defined in the BAROC file.

2. For any event on a subnet which has either primary or secondary subnets, and where that event lies in a logic flow that it connects to those subnets, it will have its ancestral_class slot populated with a list of all Subnet_Ancessor names for all classes which are primary to it and lie on the logical flows. The converse situation is used for Subnet_Descendant to populate the descendant_class slot.
3. Path_ID slot is propagated with the list of all the path numbers that the event is relevant to within its own ERN. An example is illustrated in Fig. 6 and described in the discussion on the propagation of path ID numbers below.
4. Sequence_ID slot is filled with the appropriate sequence number for that path. An example is illustrated in Fig. 7 and described in the discussion of sequence ID numbers below.

The generic correlation template becomes:

- Step 1: For any event of class within Subnet_Ancessor;
if present, make the current event secondary, and remove from future correlational analysis.
- Step 2: For any event of class within Subnet_Descendant,
if present, make the current event the primary event, and remove the older event from future correlational analysis.
- Step 3: For any event on the subnet (Subnet_Class) where the Path_ID of the current event intersects the Path_ID of the older event;

if the current event sequence_ID < older event sequence_ID, make the current event primary and remove the older event from future correlational analysis;
if the current event sequence_ID > older event sequence_ID, remove the current event from future correlational analysis, as it is a secondary event.

5 Step 4: If all of the above conditions fail, then de facto this is the primary event.

This describes the most general case for analysis. If the event had been identified in the EMD design as a true primary event, then step 1, and the second half of step 3, can be removed from the analysis, since by definition a true primary event can never be secondary to anything else. Similarly, for a true secondary event, step 2, and the first half of step 3, can be removed since a
10 secondary event can never have any events following it.

All templates can also include conditions for duplicate detection, trouble ticket forwarding, scripts to be fired, etc. These conditions are determined through a final workshop with the appropriate subject matter experts (SMEs) and follow a template similar to the document in the appendix.

15 The propagation of path ID numbers is as follows:

Step 1: Path ID numbers are contained in the set $\{1, 2, n+1...m\}$.

Step 2: Starting at the root event of the ERN (either a primary or primary/secondary event) proceed to the first nodal event. The path leading in by default is path
1. At the nodal event, one path leading out continues to be the inherited path coming in. The remaining paths are assigned an ascending sequence of path
20 IDs.

Step 3: Proceed to the next nodal event on any path and repeat the procedure. Note that no path ID number can be reused. You must always assign the next path number in the sequence.

Step 4: Back propagate the paths. For example, consider a simple two branch system. The events on path 1 that are precedent to the nodal event where path 2 arises are, by definition, also members of path 2. Therefore their path_id slot must contain both numbers in its list.

Fig. 6 illustrates the complex correlation diagram of Fig. 3 with path ID numbers added.

The propagation of sequence ID numbers is as follows:

Step 1: Sequence ID numbers are contained in the set $\{1, 2, n+1 \dots m\}$. Each event has one unique ID number.

Step 2: Starting at the root event for the ERN, assign the sequence ID numbers down one complete path. Return to the first nodal event, and continue to assign the numbers for that path, increasing from the current sequence ID of that nodal event. For example, if the nodal event has a sequence ID of 3, then the next element down any path must be 4. This process is repeated until complete.

Step 3: The only deviation occurs if one event is fed by two paths. At this point, that event should take the greater of the two values, and any events that follow it in the logical sequence should be increased accordingly. It is not required that the sequence ID's be a perfect $n+1$ increment, only that the greater than/less than relationships be maintained along any path.

Fig. 7 illustrates the complex correlation diagram of Fig. 3 with path ID and sequence ID numbers added.

The procedures for automated rule generation extensions to the methodology described in co-pending patent application, serial number 09/488,689 are as follows:

- 5 Step 1: The current methodology propagates a BAROC page using the event names as class names. It separates the autonomous and correlation candidate events. This facility needs to be retained for autonomous events. The BAROC file for correlated events is generated from the Visio drawings, propagating the appropriate slots.
- 10 Step 2: Meta_classes need to be named as per the defined convention.
- Step 3: Visio diagrams currently verify the ERN for proper use of connectors, shapes, and subnet links. This process should be repeated.
- Step 4: The Visio program then prompts for default values related to trouble ticketing, duplicate detection, status and severity changes, time to search the event cache, and locations of scripts to be executed. These values come from the final workshop with the subject matter experts (SMEs). It also requests the location of the templates to be used.
- 15 Step 5: The Visio program then defines the path and sequence ID numbers, and back propagates them into the appropriate parts of the BAROC file.

Step 6: The Visio program identifies primary and secondary subnets, and defines the ancestor and descendant classes as appropriate, and propagates those names into the respective slots on the BAROC file.

Step 7: The Visio program then generates the automated rules based on the assigned values and templates.

Step 8: The spreadsheet generates the BAROC files from the populated spreadsheet.

Step 9: The rule set and BAROC files are given a reality check by the implementor. If everything appears adequate, they are loaded for testing.

Fig. 8 illustrates the processing logic for the generation of ERNs that include BAROC classes. This is an extension of the processing logic for the generation of ERNs described in co-pending patent application, serial number 09/488,689, previously incorporated by reference. Processing starts in logic block 800 with the completion of event management questionnaire 820. The completed questionnaire is then used to customize and update rule templates for each event type within the event source as indicated in logic block 802. Customizing or updating the rule templates in logic block 802 includes adding, modifying and deleting rule actions and commands to perform the event management behavior desired by a particular customer. Once the behavior is specified, the tool fills in event names, subnet names, and related values as script names.

In logic block 804, the drawing files with ERNs are examined to verify that the ERNs do not violate any protocols. This is followed in decision block 808 with a determination as to whether or not the ERNs in the generated ERN verification file have been successfully verified. If unsuccessful, the ERNs are fixed as indicated in logic block 806 before going through re-

verification. The ERNs 850 are then used in logic block 810 to generate BAROC classes (using the processing logic illustrated in Fig. 9) for each event source. Next, in logic block 812, rules are generated for each event type within the event source automatically from the ERNs, rule templates, and default property files 860. The solution is then deployed and monitored as indicated in logic block 814.

Fig. 9 illustrates the processing logic for the automated generation of BAROC files from ERNs, corresponding to logic block 810 in Fig. 8. Processing starts in logic block 900 which indicates that events on a subnet are assigned to a superclass with the name of the subnet. In logic block 902, connected events that are on the same subnet are assigned a path number. This step is followed in decision block 904 with a test for whether or not a branch point has been reached. If it has, then a new path number is assigned to the branch and the path number is added to events earlier in the sequence than the branch point. This list is then appended to the path_id slot for each event. These acts are indicated in logic block 906. If a branch point has not been reached, as tested for in decision block 904, then incrementing sequence numbers are assigned to each event and added to the sequence_id list as indicated in logic block 908.

The next step in the processing logic of Fig. 9 is the test for a prior connected subnet in decision block 910. If there is a prior connected subnet, then in logic block 912, the connected events on this subnet that lead to the subsequent subnet are determined, this subset of events is assigned to a superclass of Subnet_Ancessor, and the superclass is added to the ancestor slot of the original subnet events. If there is no prior connected subnet, processing continues as indicated in decision block 914 with a determination of whether or not there is a subsequent connected subnet.

If there is subsequent connected subnet, then the acts indicated in logic block 916 are performed. These include a determination of the connected events on this subnet that follow from the prior subnet. This subset of events is assigned to a superclass of Subnet_Descendant, and the superclass is added to the descendant slot of the original subnet events. If there are no subsequent connected subnets in decision block 914, the BAROC classes are written to a file as indicated in logic block 918.

The following is a template for a primary/secondary event. Elements enclosed in { } are variables that are assigned by the subject matter expert (SME) for implementation. As such, changes in severity, status, administrators, etc. are then automatically filled in by the tool to create a working rule.

```
rule: '{ernname}_primary_secondary_event':  
(  
  /* Instantiate the required variable to perform correlation analysis within the event.  
  * The determination of the state PS defines this rule as specific for primary/secondary events  
  */  
  
  event: _ps_event of_class '{ernname}_Event'  
  
  where [state: equals 'PS',  
  
  hostname: _hostname,  
  
  date: _date,  
  
  subnet: _subnet,  
  
  path id: _path_id,
```

END9-2000-047US1

sequence_id: _sequence_id,

ancestor: _ancestor,

descendant: _descendant,

clearpath: _clearpath],

5 /* This is a conditional test if a clearing event, due to network variations, has arrived prior to the
 * event that it clear.

 * This also demonstrates the simple correlation algorithm for clearing events, and could be used
 * as the basis for a clearing rule.

 *

10 * The specification of the subnet equality defines that the event is one of a set of clearing events
 * that could possibly occur within a given subnet. The path_id, is required for more specific
 * correlation.

 *

15 * The intersection of the path_id and clearpath is sufficient to define the complete clearing
 * correlation.

 */

reception_action: 'Check_Clearing_Event':

(

 first_instance (event: _clr_event of_class '{ernname}_Event'

20 where [subnet: equals _subnet

```

state: equals 'C',

origin: equals _origin,

hostname: equals _hostname,

status: within ['ACK', 'OPEN'],

5   path_id: _clr_path_id

    ),

    intersect(_clearpath, _clr_path_id),

    set_event_status(_ps_event, {statusl}),

    set_event_severity(_ps_event, {severityl}),

10   set_event_administrator(_ps_event, {administrator}),

    /*

    *   exec_prog(notification of clear),

    */

    commit_set

15   ),

/* This action is a correlation to determine if the arriving event is a secondary event to an existing

* root cause that may exist on another subnet. The entire correlation is defined in the call to search

* the cache for an event within the metaclasses defined in the ancestor slot.

*/

20   reception_action: 'Check_ancestor_Subnets':

```

END9-2000-047US1

```

(
    first_instance(event: _anc_event of_class within_ancestor
        where [severity: outside ['HARMLESS'],
            status: within ['ACK', 'OPEN']]),
5    set_event_status(_ps_event, {status2}),
    set_event_severity(_ps_event, {severity1}),
    set_event_administrator(_ps_event, {administrator}),
    link_effect_to_cause( ps_event, _anc_event),
/*
10    *    exec_prog("notification of secondary status")
    */

    commit_set

),

/* This action determines if the event under examination is secondary to events on the same subnet
15    * as itself.
    * The subnet matching conditions are defined by the equality check on the subnet slot.
    *
    * The correlation occurs in two statements. The intersection of the list of paths determines if there
    * is one path in common between the two events. If this condition is satisfied, then the success of
20    * the relationships between the sequence id's will define the existing event as a secondary one to
    * an already exiting root cause.

```

END9-2000-047US1

*/

action: 'Check_ERN_Relations_Secondary':

(

first_instance(event: _related_event of _class '{ernname)_Event'

5

where [origin: equals _origin,

hostname: equals _hostname,

subnet: equals _subnet,

state: within ['PS', 'P'],

path_id: rel_path_id,

10

sequence_id: relsequence_id outside [_sequence_id],

status: within ['ACK', 'OPEN']]),

intersect(_path_id, _rel_path_id),

_sequence_id > _rel_sequence_id,

set_event_status(_ps_event, {status2}),

15

set_event_severity(_ps_event, (severity1)),

set_event_administrator(_ps_event, {administrator}),

link_effect_to_cause(_ps_event, _related_event),

/*

* exec_prog(notification of secondary status),

END9-2000-047US1

*/

commit_set

),

/* This action is a correlation to determine if the arriving event is a primary event to an existing root

5 * cause event that may exist on another subnet. The entire correlation is defined in the call to search

* the cache for an event within the metaclasses defined in the descendant slot.

*/

reception_action: 'Check_Descendant_Subnets':

(

10 first_instance(event: _desc_event of _class_within _descendant

where [severity: outside ['HARMLESS'],

status: within ['ACK', 'OPEN']]),

set_event_status(_desc_event, {status2})

set_event_severity(_desc_event, {severity1}),

15 set_event_administrator(_desc_event, {administrator}),

link_effect_to_cause(_desc_event, _ps_event)

/*

* exec_prog("notification of root cause status")

),

END9-2000-047US1

*/ This action determines if the event under examination is primary to events on the same subnet
* as itself.

* The subnet matching conditions are defined by the equality check on the subnet slot.

*

5 * The correlation occurs in two statements. The intersection of the list of paths determines if there
* is one path in common between the two events. If this condition is satisfied, then the success of
* the relationships between the sequence id's will define the existing event as a primary one to an
* already existing root cause.

*/

10 reception_action: 'Check_ERN_Relations_Primary':

(

first_instance(event: _related_event of_class '{ernname}_Event'

where [origin: equals _origin,

hostname: equals _hostname,

15 subnet: equals _subnet,

state: within ['PS', 'S'],

path_id: _rel_path_id,

sequence_id: _rel_sequence_id outside [_sequence_id],

status: within ['ACK', 'OPEN']]),

20 intersect(_path_id, _rel_path_id),

```

        _sequence_id < _rel_sequence_id,

        set_event_status(_related_event, {status2}),

        set_event_severity(_related_event, {severity1}),

        set_event_administrator(_related_event, {administrator}),

5      link_effect_to_cause(_related_event, _ps_event)

      /*

      *          exec_prog("notification of root cause status")

      */

      ),

10  /* If all of the above conditions have failed, then we may assume that the event arriving is,

      * in effect, a root cause event at this time. As such, we could notify immediately on this event.

      */

      reception_action: 'notify_on_primary':

      (

15  /*

      *          exec_prog("notification of root cause status"),

      */

      commit_set

      )

20  ).

```

END9-2000-047US1

Obviously, there are any number of actions that can be taken at different levels. This is only one possible example, and is set by the policy workshop with the SMEs. A further implication is that the rules for primary events contain only the lower half of this rule, while secondary events contain the upper half. In a uniform environment, where all event sources are handled in an identical fashion and all possible types of events exist, there are a total of six rule sets: duplicate detection and trouble ticketing; autonomous events; primary events; primary/secondary events; secondary events; and clearing events.

The current state of the EMD-IT tool, while automating certain aspects of implementing an Event Management Design, has two limitations. The implementor is still required to create the ruleset templates to handle correlation, and to generate a rule for every type of event (primary, primary/secondary, secondary, and clearing event) for each subnet within the ERN. As such, there are a large number of rules generated, and the implementor is still required to develop the bulk of the code to do correlation. Also, while there is facility to generate BAROC files, this is a manual procedure to establish any hierarchical information.

This methodology provides a template with all the sufficient code to handle correlational analysis. The implementor only needs to add specific actions after the correlation is complete. These actions are comparatively simple and require little training to implement. Because of the nature of this type of correlation, only four rules are required for the entire ERN, rather than four rules for each rule on the ERN. Since the structure of the BAROC files in a well-mannered hierarchy is requisite for this model, BAROC file generation is now completely automated.

The present invention can be realized in software or a combination of hardware and software. Any kind of computer system or other apparatus adapted for carrying out the methods described herein is suited. A typical combination of hardware and software could be a general purpose computer system with a computer program that, when loaded and executed, controls the computer system such that it carries out the methods described herein. The present invention can also be embedded in a computer program product, which includes all the features enabling the implementation of the methods described herein, and which, when loaded in a computer system, is able to carry out these methods.

Computer program instructions or computer program in the present context means any expression, in any language, code or notation, or a set of instructions intended to cause a system having an information processing capability to perform a particular function, either directly or when either or both of the following occur: a) conversion to another language, code or notation; b) reproduction in a different material form.

Those skilled in the art will appreciate that many modifications to the preferred embodiment of the present invention are possible without departing from the spirit and scope of the present invention. In addition, it is possible to use some of the features of the present invention without the corresponding use of the other features. Accordingly, the foregoing description of the preferred embodiment is provided for purpose of illustrating the principles of the present invention and not in limitation thereof, since the scope of the present invention is defined solely by the appended claims.

END9-2000-047US1